**M矩阵以列为主序**

bool Mat4::decompose(Vec3\* scale, Quaternion\* rotation, Vec3\* translation) const

{

if (translation)

{

// Extract the translation.

translation->x = m[12];

translation->y = m[13];

translation->z = m[14];

}

// Nothing left to do.

if (scale == nullptr && rotation == nullptr)

return true;

// Extract the scale.

// This is simply the length of each axis (row/column) in the matrix.

Vec3 xaxis(m[0], m[1], m[2]);

float scaleX = xaxis.length();

Vec3 yaxis(m[4], m[5], m[6]);

float scaleY = yaxis.length();

Vec3 zaxis(m[8], m[9], m[10]);

float scaleZ = zaxis.length();

// Determine if we have a negative scale (true if determinant is less than zero).

// In this case, we simply negate a single axis of the scale.

float det = determinant();

if (det < 0)

scaleZ = -scaleZ;

if (scale)

{

scale->x = scaleX;

scale->y = scaleY;

scale->z = scaleZ;

}

// Nothing left to do.

if (rotation == nullptr)

return true;

// Scale too close to zero, can't decompose rotation.

if (scaleX < MATH\_TOLERANCE || scaleY < MATH\_TOLERANCE || fabs(scaleZ) < MATH\_TOLERANCE)

return false;

float rn;

// Factor the scale out of the matrix axes.

rn = 1.0f / scaleX;

xaxis.x \*= rn;

xaxis.y \*= rn;

xaxis.z \*= rn;

rn = 1.0f / scaleY;

yaxis.x \*= rn;

yaxis.y \*= rn;

yaxis.z \*= rn;

rn = 1.0f / scaleZ;

zaxis.x \*= rn;

zaxis.y \*= rn;

zaxis.z \*= rn;

// Now calculate the rotation from the resulting matrix (axes).

float trace = xaxis.x + yaxis.y + zaxis.z + 1.0f;

if (trace > MATH\_EPSILON)

{

float s = 0.5f / sqrt(trace);

rotation->w = 0.25f / s;

rotation->x = (yaxis.z - zaxis.y) \* s;

rotation->y = (zaxis.x - xaxis.z) \* s;

rotation->z = (xaxis.y - yaxis.x) \* s;

}

else

{

// Note: since xaxis, yaxis, and zaxis are normalized,

// we will never divide by zero in the code below.

if (xaxis.x > yaxis.y && xaxis.x > zaxis.z)

{

float s = 0.5f / sqrt(1.0f + xaxis.x - yaxis.y - zaxis.z);

rotation->w = (yaxis.z - zaxis.y) \* s;

rotation->x = 0.25f / s;

rotation->y = (yaxis.x + xaxis.y) \* s;

rotation->z = (zaxis.x + xaxis.z) \* s;

}

else if (yaxis.y > zaxis.z)

{

float s = 0.5f / sqrt(1.0f + yaxis.y - xaxis.x - zaxis.z);

rotation->w = (zaxis.x - xaxis.z) \* s;

rotation->x = (yaxis.x + xaxis.y) \* s;

rotation->y = 0.25f / s;

rotation->z = (zaxis.y + yaxis.z) \* s;

}

else

{

float s = 0.5f / sqrt(1.0f + zaxis.z - xaxis.x - yaxis.y );

rotation->w = (xaxis.y - yaxis.x ) \* s;

rotation->x = (zaxis.x + xaxis.z ) \* s;

rotation->y = (zaxis.y + yaxis.z ) \* s;

rotation->z = 0.25f / s;

}

}

return true;

}